MAPSI

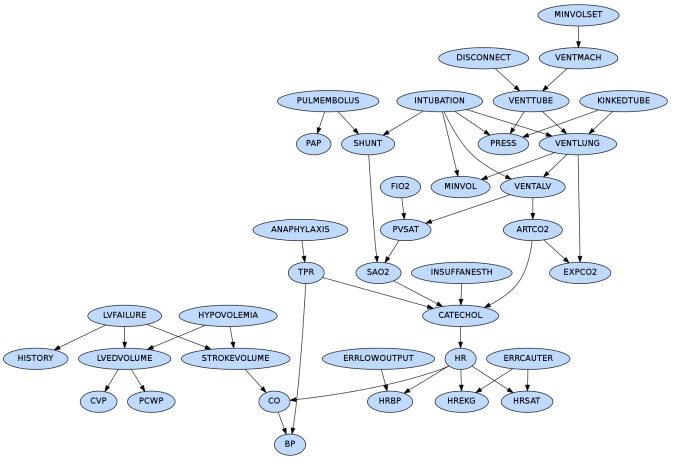
Modèles et Algorithmes de Probabilités et Statistiques pour l'Informatique

Indépendances conditionnelles et réseaux bayésiens

Dans ce TME, l'objectif est d'apprendre des réseaux bayésiens à partir de bases de données. Hormis la base **asia**, un exemple jouet relativement petit qui vous permettra de mettre au point les différents algorithmes du TME, et **car**, les autres bases correspondront à des distributions de probabilité de tailles raisonnables :

nom de la base	provenance	nombre d'événements élémentaires
asia	BN repository	256
alarm	BN repository	10^{16}
adult	UCI machine learning repository	10^{12}
car	UCI machine learning repository	6912
agaricus-lepiota	UCI machine learning repository	10^{16}

Apprendre un réseau bayésien consiste à apprendre sa structure graphique ainsi que les paramètres de ses distributions de probabilité conditionnelles. Pour réaliser la deuxième tâche, il suffit d'estimer les paramètres de chaque distribution conditionnelle par maximum de vraisemblance, comme vous l'avez fait dans le TME 3. Ici, nous nous focaliserons donc plutôt sur l'apprentissage de structure. Celle-ci reflétant des indépendances conditionnelles entre variables aléatoires, vous devrez exploiter des tests d'indépendance du χ^2 afin d'obtenir des structures graphiques les moins denses possibles (en termes de nombres d'arcs). Ainsi, **alarm** représente une distribution jointe de plus de 10^{16} événements élémentaires mais, quand cette distribution est décomposée grâce au graphe ci-dessous, elle peut être décrite (sans perte d'informations) à l'aide de seulement 752 paramètres. Comme nous l'avons vu en cours, cette représentation permet également d'effectuer très rapidement des calculs probabilistes.



1. Lecture des données

Dans le code ci-dessous, la fonction **read_csv** vous permettra de lire les données des bases sur lesquelles vous allez travailler, et de les organiser sous une forme adéquate. La fonction **read_csv** prend en argument le nom d'un fichier CSV et renvoie un triplet :

- 1 tableau de strings contenant les noms des variables aléatoires
- un tableau 2D contenant les données du fichier CSV encodées sous forme numérique (les valeurs des variables aléatoires sont transformées en nombres entiers)
- un tableau de dictionnaires faisant la correspondance, pour chaque variable aléatoire, entre l'encodage numérique et les données du fichier CSV (le 1er dictionnaire correspond à la variable de la 1ère colonne du CSV, le 2ème dictionnaire à celle de la 2ème colonne, *etc.*)

Téléchargez le fichier **asia dataset** et lisez-le à l'aide de la fonction **read_csv**.

```
# -*- coding: utf-8 -*-

import numpy as np

# fonction pour transformer les données brutes en nombres de 0 à n-1
def translate_data ( data ):
    # création des structures de données à retourner
    nb_variables = data.shape[0]
    nb_observations = data.shape[1] - 1 # - nom variable
    res_data = np.zeros ( (nb_variables, nb_observations ), int )
    res_dico = np.empty ( nb_variables, dtype=object )

    # pour chaque variable, faire la traduction
    for i in range ( nb_variables ):
        res_dico[i] = {}
        index = 0
        for j in range ( 1, nb_observations + 1 ):
            # si l'observation n'existe pas dans le dictionnaire, la rajouter
            if data[i,j] not in res_dico[i]:
                res_dico[i].update ( { data[i,j] : index } )
                index += 1
            # rajouter la traduction dans le tableau de données à retourner
            res_data[i,j-1] = res_dico[i][data[i,j]]
    return ( res_data, res_dico )


# fonction pour lire les données de la base d'apprentissage
```

SEARCH

Go

TUTORIEL NUMPY

INFOS COURS/TD/TME

SEMAINIER

Semaine 1
Semaine 2
Semaine 3
Semaine 4
Semaine 5
Semaine 6?
Semaine 7?
Semaine 8
Semaine 9?
Semaine 10?

LIENS

edit SideBar

```
def read_csv ( filename ):
    data = np.loadtxt ( filename, delimiter=',', dtype='string' ).T
    names = data[:,0].copy ()
    data, dico = translate_data ( data )
    return names, data, dico
```

[$[Get Code]]

2. Statistique du χ^2 conditionnel

Soit deux variables aléatoires X et Y. Appelons N_{xy}, N_x et N_y, respectivement, le nombre d'occurrences du couple $(X=x, Y=y)$ et des singletons $X=x$ et $Y=y$ dans la base de données. Alors, comme indiqué sur la planche 37 du cours 5, la statistique du χ^2 de X et Y est égale à :

$$\chi^2_{X,Y} = \sum_x \sum_y \frac{\left(N_{xy} - \frac{N_x \times N_y}{N}\right)^2}{\frac{N_x \times N_y}{N}}$$

où N représente le nombre de lignes de la base de données. Cette formule permet de tester l'indépendance entre les deux variables X et Y. On peut aisément généraliser celle-ci pour tester des indépendances conditionnellement à un ensemble de variables $\mathbf{Z}$:

$$\chi^2_{X,Y|\mathbf{Z}} = \sum_x \sum_y \sum_{\mathbf{z}} \frac{\left(N_{xy\mathbf{z}} - \frac{N_{x\mathbf{z}} \times N_{y\mathbf{z}}}{N}\right)^2}{\frac{N_{x\mathbf{z}} \times N_{y\mathbf{z}}}{N_{\mathbf{z}}}}$$

où $N_{xy\mathbf{z}}$, $N_{x\mathbf{z}}$, $N_{y\mathbf{z}}$ et $N_{\mathbf{z}}$ représentent, respectivement, le nombre d'occurrences du triplet $(X=x, Y=y, \mathbf{Z}=\mathbf{z})$, des couples $(X=x, \mathbf{Z}=\mathbf{z})$ et $(Y=y, \mathbf{Z}=\mathbf{z})$, et du singleton $\mathbf{Z}=\mathbf{z}$. Ainsi, si $\mathbf{Z}$ est un ensemble de 3 variables aléatoires (A, B, C), les valeurs $\mathbf{z}$ seront des triplets (a, b, c).

Ecrivez une fonction **sufficient_statistics** (data, dico, x, y, z) qui, étant donné le tableau 2D des données et le dictionnaire obtenus à la question 1, ainsi que deux entiers x et y représentant deux variables aléatoires (les index de leurs colonnes dans le CSV, en commençant à l'index 0 pour la 1ère colonne) et une liste d'entiers z, potentiellement vide, représentant la liste des variables de $\mathbf{Z}$ (index de colonnes dans le CSV), renvoie la valeur de $\chi^2_{X,Y|\mathbf{Z}}$. Ainsi, **sufficient_statistics** (data, dico, 0, 1, [2,3]) renverra la statistique pour X = la variable de la 1ère colonne du CSV, Y = la variable aléatoire de la 2ème colonne du CSV, et $\mathbf{Z}$ = le couple des variables des 3ème et 4ème colonnes du CSV. Pour vous éviter de parser le tableau data afin de compter les nombres $N_{xy\mathbf{z}}$, $N_{x\mathbf{z}}$, $N_{y\mathbf{z}}$ et $N_{\mathbf{z}}$, vous pourrez utiliser la fonction **create_contingency_table** (data, dico, x, y, z) ci-dessous qui, étant donné les mêmes arguments que votre fonction **sufficient_statistics**, renvoie un tableau de couples $(N_{\mathbf{z}}, T_{X,Y})$, pour tous les $\mathbf{z} \in \mathbf{Z}$, où $T_{X,Y}$ est un tableau 2D (première dimension en X et deuxième en Y) contenant les $N_{x,y,\mathbf{z}}$. Par exemple, **create_contingency_table** (data, dico, 1, 2, [3]) peut renvoyer le tableau suivant :

```
array([(538,  array([[ 497.,    1.],
                     [  40.,    0.]])),
       (9462, array([[ 4476.,  54.],
                     [ 4863.,  69.]]))])
```

ce qui correspond précisément à :

$$\begin{array}{lll} \text{array}([& (N_{Z=0}, \text{array}([& [N_{X=0,Y=0,\mathbf{Z}=0}, \quad N_{X=0,Y=1,\mathbf{Z}=0}], \\ & & [N_{X=1,Y=0,\mathbf{Z}=0}, \quad N_{X=1,Y=1,\mathbf{Z}=0}]])), \\ & (N_{Z=1}, \text{array}([& [N_{X=0,Y=0,\mathbf{Z}=1}, \quad N_{X=0,Y=1,\mathbf{Z}=1}], \\ & & [N_{X=1,Y=0,\mathbf{Z}=1}, \quad N_{X=1,Y=1,\mathbf{Z}=1}]]))]) \end{array}$$

Pour un $\mathbf{z}$ donné, calculer $N_{x\mathbf{z}}$ revient donc à faire des sommes sur chaque ligne des $N_{xy\mathbf{z}}$ et calculer les $N_{y\mathbf{z}}$ revient à faire des sommes en colonne. **Attention :** il peut arriver que certains $N_{\mathbf{z}}$ soient égaux à 0. Dans ce cas, vous ne tiendrez pas compte des $N_{xy\mathbf{z}}$, $N_{x\mathbf{z}}$ et $N_{y\mathbf{z}}$ correspondants dans la formule de $\chi^2_{X,Y|\mathbf{Z}}$ (car vous feriez des divisions par 0, ce qui est mal).

```
# etant donné une BD data et son dictionnaire, cette fonction crée le
# tableau de contingence de (x,y) | z
def create_contingency_table ( data, dico, x, y, z ):
    # détermination de la taille de z
    size_z = 1
    offset_z = np.zeros ( len ( z ) )
    j = 0
    for i in z:
        offset_z[j] = size_z
        size_z *= len ( dico[i] )
        j += 1

    # création du tableau de contingence
    res = np.zeros ( size_z, dtype = object )

    # remplissage du tableau de contingence
    if size_z != 1:
        z_values = np.apply_along_axis ( lambda val_z : val_z.dot ( offset_z ),
                                         1, data[z,:].T )
        i = 0
        while i < size_z:
            indices, = np.where ( z_values == i )
            a,b,c = np.histogram2d ( data[x,indices], data[y,indices],
                                     bins = [ len ( dico[x] ), len (dico[y] ) ] )
            res[i] = ( indices.size, a )
            i += 1
    else:
        a,b,c = np.histogram2d ( data[x,:], data[y,:],
                                 bins = [ len ( dico[x] ), len (dico[y] ) ] )
        res[0] = ( data.shape[1], a )
    return res
```

[$[Get Code]]

3. Statistique du χ^2 et degré de liberté

Modifiez votre fonction **sufficient_statistics** afin qu'elle ne renvoie plus seulement $\chi^2_{X,Y|\mathbf{Z}}$ mais plutôt un couple ($\chi^2_{X,Y|\mathbf{Z}}$,DoF), où DoF est le nombre de degrés de liberté de votre statistique. Celui-ci est égal à :

$$(|X| - 1) \times (|Y| - 1) \times |\{\mathbf{z} : N_{\mathbf{z}} \neq 0\}|$$

où $|X|$ représente le nombre de valeurs possibles que peut prendre la variable X, autrement dit, c'est la taille de

son dictionnaire. Le dernier terme de l'équation est simplement le nombre de $N_{\mathbf{z}}$ différents de 0.

4. Test d'indépendance

En cours, nous avons vu que, pour un risque α donné, si la statistique $\chi^2_{X,Y|\mathbf{Z}}$ est inférieure au seuil critique c_α de la loi du χ^2 à DoF degrés de liberté, alors X et Y sont considérés comme indépendants conditionnellement à $\mathbf{Z}$ ($X \perp\!\!\!\perp Y|\mathbf{Z}$). On peut reformuler cette propriété de la manière suivante :

$$\text{p-value}(\chi^2_{X,Y|\mathbf{Z}}) \geq \alpha \Longleftrightarrow X \perp\!\!\!\perp Y|\mathbf{Z}$$

La p-value d'un nombre x est l'intégrale de la fonction de densité de la loi du χ^2 de x à $+\infty$ (autrement dit, c'est la surface de la partie grisée sur votre table du χ^2 à partir de l'abscisse x. On a donc p-value$(c_\alpha) = \alpha$). En statistiques, on considère qu'elle n'a du sens que si les valeurs du tableau de contingence sont toutes supérieures ou égales à 5 (autrement dit, un test d'indépendance du χ^2 n'est "valide" que si toutes les valeurs du tableau de contingence sont supérieures ou égales à 5). En informatique, on allège souvent cette règle en considérant que le test est valide dès lors que la valeur moyenne des cases est supérieure ou égale à 5. Cet allègement permet de tester la validité du test sans réaliser celui-ci : si le nombre de lignes du CSV est supérieure ou égale à $d_{min} = 5 \times |X| \times |Y| \times |\mathbf{Z}|$, le test est considéré comme valide.

Ecrivez une fonction **indep_score**(data, dico, x, y, z) qui, étant donné les mêmes paramètres que ceux de la question précédente, vous renvoie la p-value correspondant à $\chi^2_{X,Y|\mathbf{Z}}$. Vous testerez au préalable si len (data[0]), le nombre de lignes de votre CSV, est supérieur ou non à d_{min} ; si c'est inférieur, vous renverrez le couple (-1,1), qui représente une indépendance. Vous pourrez vous aider de la fonction scipy.stats.chi2.sf (x, DoF) qui renvoie la p-value (x) pour une loi à DoF degrés de liberté.

```
import scipy.stats as stats
stats.chi2.sf ( x, DoF )
```

[$[Get Code]]

5. Meilleur candidat pour être un parent

Ecrivez une fonction **best_candidate** (data, dico, x, z, alpha) qui, étant donné une variable aléatoire X et un ensemble de variables aléatoires $\mathbf{Z}$ détermine la variable Y (en fait, l'index de sa colonne dans le CSV), parmi toutes celles à gauche de X dans le fichier CSV, qui est la plus dépendante de X conditionnellement à $\mathbf{Z}$, autrement dit, celle qui a la plus petite p-value. Si cette p-value est supérieure à alpha, cela veut dire que $\chi^2_{X,Y|\mathbf{Z}}$ est inférieur à c_α et donc que Y est jugée indépendante de X conditionnellement à $\mathbf{Z}$. Votre fonction renverra une liste vide si Y est indépendante de X conditionnellement à $\mathbf{Z}$, sinon elle renverra une liste contenant Y. Vous pourrez tester votre fonction avec α = 0.05.

6. Création des parents d'un noeud

Ecrivez une fonction **create_parents** (data, dico, x, alpha) qui, étant donné une variable aléatoire x et un niveau de risque alpha, retourne la liste z de ses parents dans le réseau bayésien. L'algorithme est le suivant : partez de z = l'ensemble vide, puis tant que **best_candidate** (x, z, alpha) vous renvoie une liste non vide [y], rajoutez y à z. Lorsque vous sortirez de cette boucle, toutes les autres variables seront indépendantes de x conditionnellement à z.

L'algorithme qui consiste à appliquer, pour chaque noeud/variable aléatoire, votre fonction **create_parents** correspond, en grande partie, à l'article suivant :

Gregory F. Cooper and Edward Herskovits (1992) "A Bayesian method for the induction of probabilistic networks from data", *Machine Learning*, Vol. 9, n°4, pp. 309-347.

7. Apprentissage de la structure d'un réseau bayésien

Ecrivez une fonction **learn_BN_structure** (data, dico, alpha) qui renvoie un tableau contenant, pour chaque noeud, la liste de ses parents. Ainsi, si votre fonction vous renvoie le tableau ci-dessous,

```
array( [ [], [], [0], [1], [1], [2, 3], [4, 5], [3, 2] ] )
```

les noeud correspondant aux 2 premières colonnes du CSV n'ont pas de parents, le noeud de la 3ème colonne a pour parent celui de la 1ère colonne, etc.

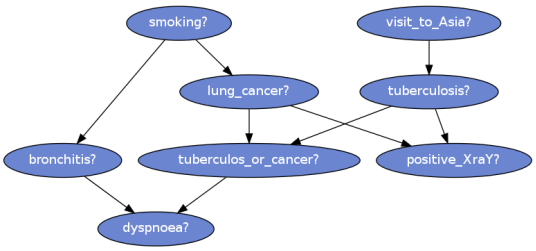
Pour visualiser plus aisément votre structure, utilisez la fonction **display_BN** ci-dessous. Celle-ci prend en paramètres :

1. le tableau des noms des variables aléatoires déterminé à la question 1
2. la structure que vous avez calculée avec votre fonction **learn_BN_structure**
3. un nom que vous voulez donner à votre réseau
4. un style pour afficher les noeuds

```
import pydot
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

style = { "bgcolor" : "#6b85d1", "fgcolor" : "#FFFFFF" }

def display_BN ( node_names, bn_struct, bn_name, style ):
    graph = pydot.Dot( bn_name, graph_type='digraph')

    # création des noeuds du réseau
    for name in node_names:
        new_node = pydot.Node( name,
                               style="filled",
                               fillcolor=style["bgcolor"],
                               fontcolor=style["fgcolor"] )
        graph.add_node( new_node )

    # création des arcs
    for node in range ( len ( node_names ) ):
        parents = bn_struct[node]
        for par in parents:
            new_edge = pydot.Edge ( node_names[par], node_names[node] )
            graph.add_edge ( new_edge )

    # sauvegarde et affaichage
    outfile = bn_name + '.png'
    graph.write_png( outfile )
```

```
    img = mpimg.imread ( outfile )
    plt.imshow( img )
```

[$[Get Code]]



7. Fin de l'apprentissage et calcul probabiliste

Comme précisé au début du TME, apprendre un réseau bayésien consiste à déterminer sa structure graphique et estimer ses paramètres. Vous avez réalisé la première partie. La deuxième, plus simple, peut se faire par maximum de vraisemblance pour chaque table de probabilité des noeuds conditionnellement à leurs parents, comme dans le TME 3. Utilisez la fonction **learn_parameters** ci-dessous pour effectuer cette tâche. Cette fonction prend en paramètres la structure graphique que vous avez apprise ainsi que le nom du fichier CSV que vous avez utilisé pour votre apprentissage. Elle renvoie un réseau bayésien à la **aGrUM**. Pour pouvoir utiliser aGrUM, reportez-vous à la **question 7 du TME 2**.

```
import pyAgrum as gum
import gumLib.notebook as gnb
from gumLib.pretty_print import pretty_cpt


def learn_parameters ( bn_struct, ficname ):
    # création du dag correspondant au bn_struct
    graphe = gum.DAG ()
    nodes = [ graphe.addNode () for i in range ( bn_struct.shape[0] ) ]
    for i in range ( bn_struct.shape[0] ):
        for parent in bn_struct[i]:
            graphe.addArc ( nodes[parent], nodes[i] )

    # appel au BNLearner pour apprendre les paramètres
    learner = gum.BNLearner ( ficname )
    return learner.learnParameters ( graphe )
```

[$[Get Code]]

Vous pouvez maintenant réaliser des calculs probabilistes :

- affichage de la taille du réseau bayésien

```
# création du réseau bayésien à la aGrUM
bn = learn_parameters ( bn_struct, ficname )

# affichage de sa taille
print bn
```

[$[Get Code]]

- affichage de la table de probabilité conditionnelle d'un noeud du réseau déterminé par son nom (1ère ligne du CSV):

```
# récupération de la ''conditional probability table'' (CPT) et affichage de cette table
pretty_cpt ( bn.cpt ( bn.idFromName ( 'bronchitis?' ) ) )
```

[$[Get Code]]

- calcul de la probabilité marginale d'un noeud : P('bronchitis?'):

```
# calcul de la marginale
proba = gnb.getPosterior ( bn, {}, 'bronchitis?' )

# affichage de la marginale
pretty_cpt ( proba )
```

[$[Get Code]]

- affichage graphique d'une distribution de probabilité marginale

```
gnb.showPosterior ( bn, {}, 'bronchitis?' )
```

[$[Get Code]]

- calcul d'une distribution marginale a posteriori : P(bronchitis? | smoking? = true, turberculosis? = false)

```
gnb.showPosterior ( bn, {'smoking?': 'true', 'tuberculosis?' : 'false' }, 'bronchitis?' )
```

[$[Get Code]]

8. (Bonus) Autres bases de données

Vous pouvez appliquer vos algorithmes sur des bases un peu plus conséquentes qu'asia:

nom de la base	dataset	nb de lignes de la base	événements élémentaires
asia	**asia dataset**	30000	256
adult	**adult dataset**	30162	10^{12}
car	**car dataset**	1728	6912
agaricus-lepiota	**agaricus-lepiota dataset**	8124	10^{16}
alarm	**alarm dataset**	20000	10^{16}